**스펨 메일 분류 기계학습 모델 최종 보고서**

**Abstract**

본 프로젝트는, 기계학습 COSE362의 학기 텀 프로젝트로, 스팸 메일과 아닌 데이터셋을 활용하여 스팸 메일을 분류하는 Binary Classification 모델을 구현하고자 합니다. 수업 시간에 학습한 다양한 모델과 방법, 개념등에 실질적인 사용을 평가하고자 하며 이를 통해 학기중 배운 내용을 정리하고 활용하는 방향으로 이어가고자 합니다.

데이터는 스팸 메일과 스팸 메일이 아닌 두가지 메일로 나누어져 있으며, 이는 label 0과 1로 구분되어 있습니다. 이에 저는 다양한 기계학습 모델과 딥러닝 기술을 활용하여 스펨 메일 분류 모델을 만들었습니다.

머신러닝 테스크로 주로 수업시간에 배운 모델들을 활용하였습니다. 모델로는 Gaussian Naive Bayes, Multinomial Naive Bayes 등의 Naive Bayes 기반의 모델부터 시작하여, Decision Tree, RandomForest 등의 Tree 기반의 모델, 그리고 마지막으로 State-of-art 방식이자 좋은 성능을 보유한 것으로 알려진 SVM, XGBoost까지 활용하여 모델을 학습시켰습니다. 마지막으로 모델이 잘 학습되었는지 평가할 수 있는 다양한 Evaluation 평가기법으로 학습 모델을 검증하였습니다. 평가 지표로는 필수로 진행해야하는 f1 score, 그리고 그 이외에도 Accuracy, AUC, Precision\_Recall , Confusion Matrix 등을 활용하였고, 이를 시각화하여 비교가 용이롭게 되도록 진행하였습니다. 진행 결과, RandomForest의 성능이 가장 높게 나왔으며, 추가적인 RandomForest GridSearch를 활용하여 최적의 Parameter를 찾고자 하였습니다.

딥러닝 모델 역시 직접 구현해보았습니다. NLP 처리에서 강한 성능을 보여주면서 수업시간에 배운 LSTM 방식을 활용하여 딥러닝으로도 스펨 분류 메일을 만들어 보았습니다.

**Introduction**

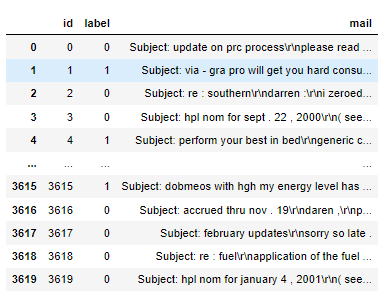
**1.1 목적, 실행환경**

해당 텀 프로젝트의 최종 목표는 스팸 분류 모델을 구현하는 것입니다. 스팸 분류 모델을 실제로 구현해보고 평가 해보면서, 수업시간에 내용을 다시한번 상기하고, 이를 활용하면서 실질적으로 사용하는 것까지 가능하게 하고자 합니다. 해당 프로젝트에는 실제로 학기중에 배운 많은 내용을 적용하면서 실행하고자 노력하였습니다.

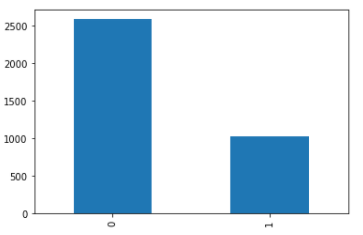
해당 프로젝트를 진행하기위해 우선, 실행환경을 세팅해주었습니다. 빠른 연산(Google GPU)를 사용할 수 있는 Google Colab, 그리고 데이터 준비 및 환경 세팅이 편리한 Jupyter Notebook중에서 고민하였는데, 우선 사용하고자 하는 기계학습 모델은 연산 시간이 오래걸리지 않은 점을 고려하여 Jupyter Notebook으로 진행하였습니다. 데이터의 형식(csv)를 고려하여 폴더를 지정하였습니다. 실행하고자 하는 main.ipynb를 기준으로 동일한 폴더에 train, test csv를 두었습니다. 또한 나오는 predict output의 csv 역시 동일한 폴더에서 진행되도록 설정하였습니다. 그리고 마지막으로 제출전에 규격에 맞도록 data/test.csv에 저장될 수 있도록 하였습니다.

**Datasets**

데이터의 Column은 총 두가지로 나누어져 있습니다. train.csv, test.csv 두 가지 모두 [‘id’],[‘mail’], [‘label’]로 이루어져 있습니다. mail 열은 실제로 우리가 구분해야 하는 mail 원본이 들어가있고, label은 이 메일이 스팸인지 아닌지 0과 1로 구분 되어 있습니다. 마지막으로 고유 id가 들어가있습니다. 특징이 있다면 원본 이메일을 그대로 복사하였기 때문에, 뛰어쓰기나 단락 변경, 주제 등의 특수문자를 포함한 기호들이 전부 포함되어 있었습니다. 결론적으로 이 mail 원문을 잘 다듬는 것이 좋은 성능의 첫번째 지표가 되었습니다.



<Figure 1. Train.csv 파일 형식>

데이터 전처리 진행 이전에, 가지고 있는 데이터의 분석을 진행하였습니다. 총 3619개의 데이터를 가지고 있었습니다. value\_counts method를 사용하여 스팸과 스팸이 아닌 데이터의 비율을 확인 해 보았습니다. 그 결과 스팸이 아닌 메일이 약 두배정도 더 많은 것을 확인하였고, 데이터 가 Imbalance함을 알 수 있었습니다. 이를 해결해주기 위해 Upsampling 등의 방식을 고려해보았지만, 달리 진행하진 않았습니다. 

<Figure 2. Label Distribution>

**Data Preprocessing**

사실상 프로젝트에서 가장 중요하다고 볼 수 있는 데이터 분석 및 전처리 과정입니다. 우리가 가지고 있는 데이터이자 구현하고자 하는 데이터는 우리가 일상에서 사용함을 뜻하는 자연어 입니다. 흔히 사용 하는 수치가 정리 되어있는 Tabular Data, Statistical Data 혹은 이미지 데이터와 다르게 자연어는 다른 전처리 과정들이 필요합니다.

특히 진행하여야 하는 전처리는 텍스트 전처리 입니다. 우리가 사용하고자 하는 이메일 데이터는 사람의 언어를 텍스트로 변환한 것으로써, 특수문자와 띄어쓰기 단락등이 포함되어 있습니다. 이러한 문자들은 모델의 학습을 저하시키고 인공지능 모델의 학습을 방해합니다. 다행히 메일이 영어이기 때문에, 띄어쓰기로 단어가 구분되는 영어의 특성상, 띄어쓰기 단어로 편하게 전처리를 진행할 수 있었습니다. 한국어의 경우 띄어쓰기 단위 안에 조사, 명사, 형용사등의 어미가 추가로 붙기 때문에 이를 활용하는 것이 비교적 어렵습니다. 이를 사용하려면 우리나라의 전처리를 도와주는 KoBERT등의 모델을 사용해주어야 합니다. 각 언어마다 사용하고 구분할 수 있는 전처리 모델이 다름을 확인하여야 합니다. 영어는 띄어쓰기 단위로 단어가 나누어져 있기 때문에, 특별히 형태소와 문장 토큰화를 진행할 필요없이 간편하게 토큰화를 진행할 수 있습니다.

**Data Cleaning**

Tokenization을 진행하기 이전에, 현재 우리가 가지고 있는 데이터는 현재 깨끗하지 않은 상태입니다. 예를 들어, 대화 맥락에 상관없은 \n, \r과 같은 문자부터, 쓰기 표현을 위한 콜론과 세미콜론, 그 이외에 특수문자 들은 학습에 있어서 아무런 의미가 없는 데이터이기에 노이즈가 됩니다. 이러한 데이터를 우선적으로 제거해주어야 합니다.

우선 전처리를 통해 불용어를 제거해주었습니다. 불용어란, 우리나라의 어미와 어사구와 같이 큰 의미를 갖지 않는 단어를 뜻합니다. 이는 데이터의 양을 늘이고 학습의 Complexity를 증가시킬 수 있고, 데이터 학습에 중요한 요소가 아니기에 제거하고자 합니다.

우리의 데이터인 영어의 경우, nltk라는 자연어 처리 라이브러리에서 영어에 대한 불용어를 확인할 수 있습니다. 우리가 가진 원본 메일 데이터에 여기서 포함시킨 불용어들을 제거한다면, 학습이 더욱 용이하게 될 것입니다. nltk에서 뽑은 불용어의 종류는 다음과 같습니다. 이는 코드에서 확인해볼 수 있습니다.

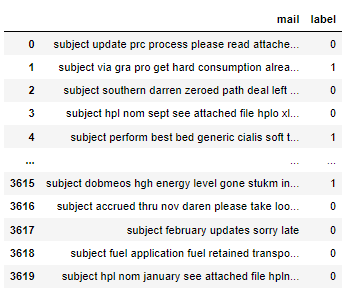
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of'’ ,... , 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

이렇게, 불용어 제거를 통해 우선적으로 데이터를 제거할 수 있었습니다.

그 다음으로, 위에서 설명했듯이, 언어에서 사용하지 않는, 텍스트로 변환하기 위해서 사용되었던 콜론과 같은 특수문자와 띄어쓰기 단위를 제거해주어야 했습니다. 이를 위해 실제로 영어 회화에서 사용되지 않는 <[\r\n]\*/?>)[^a- zA-Z0-9\s] 등을 제거해주었습니 다. 이는 실제 데이터를 살펴 보면서 포함시킨 요소도 있고, 그렇지 않고 필요없다고 생각된 특수문자를 임의로 선택하기도 하였습니다.

다음으로, 영어는 한국어와 다르게 대문자와 소문자의 개념이 있습니다. 그래서 Data, dAta, daTa, datA 등의 요소들이 전부 data라는 단어를 지칭한다는 점을 이용하였고, 추후 토큰화를 진행할 때, 이러한 대문자는 다른 단어로 인식하여 다른 벡터값을 부여하기에, 이를 모두 통일시켜줄 필요가 있었습니다. 그래서 lower()라는 method를 사용하여 대소문자 구분없이 모두 소문자로 변환시키는 과정을 진행하였습니다.

이 Data Cleaning 전처리를 통해 데이터가 좀 더 학습하기 용이한 형태로 바뀐 것을 확인할 수 있었습니다.



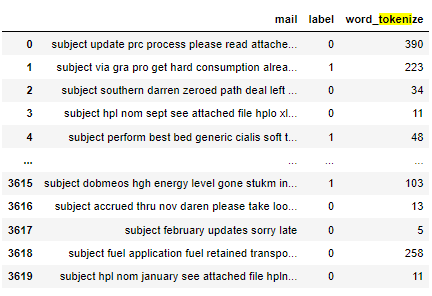
<Figure 3. 전처리 이후 데이터>

Figure 1의 전처리 이전 원본 csv 데이터와 비교해 보았을 때, 더 깔끔하고 정리되어 있는 것을 확인하실 수 있습니다. 그리고 이렇게 정리된 데이터가 띄어쓰기 단위로 단어들을 모두 지칭하고 있다는 것 또한 알 수 있습니다.

**Tokenizing**

이렇게 Figure 3과 같이 깔끔하게 전처리 된 데이터를 이용하여 토큰화를 진행해주었습니다.

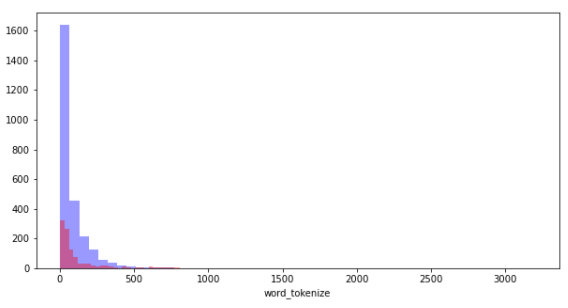
토큰화란, 주어진 corpus를 전부 token으로 생각하고, 의미를 갖는 문자열의 단위로 끊어주는 것이다. 예를 들어 I am hungry는 I, am, hungry로 나눌 수 있는 것이다. 앞서 말했듯이 나는 배가 고프다를 토큰화시키는 것은 ‘나’와 ‘는’의 차이점, ‘배’,’가’의 차이점 등 한 띄어쓰기 안에도 여러가지 단어의 차이점을 확인해야 하지만, 영어는 띄어쓰기 단위로 잘 나누어져 있기 때문에, 쉽게 토큰화를 진행할 수 있다. 또한 해당 프로젝트에서 내가 사용하고자 했던 데이터는 토큰화와 Vectorization을 자동으로 진행해주기 때문에, 처음에 토큰화의 과정을 중복해서 사용해 주었는데, 해당 오류가 발생하여 토큰화는 코드에는 포함되어 있지 않다. 하지만 데이터 분석을 위해 개별적으로 데이터 토큰화를 진행하고, 이를 칼럼에 추가해 보았다.



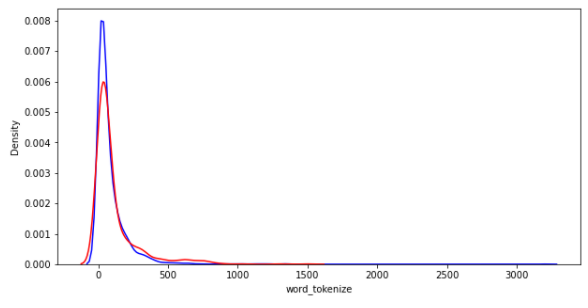
<Figure 4. After Tokenize>

이렇게 앞서 설명한 데이터 전처리를 진행한 후, 다시한번 데이터를 분석해보았다.

우선, 스팸과 스팸이 아닌 메일의 데이터 분석을 진행해보고자 하였다. 우선, 토큰화를 진행한 후에, corpus의 개수를 비교해주는 Seaborn의 distplot과 kdeplot을 그려 데이터 분포와 문장 및 단어수에 따른 구분이 가능한지 확인해보도록 하였다.



<Figure 5. Token 개수 distplot>



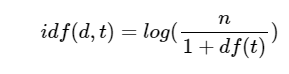
<Figure 6. Token 개수 distplot>

파란색이 스팸메일이 아닌 경우, 빨간색이 스팸인 경우를 확인해준 것인데, 현재 데이터 imbalance 때문에 정확하게 구분이 힘들긴 하지만, Kernel Density Distribution과 개수에 따른 차이는 스팸과 스팸이 아닌 경우를 구분할 수는 없다는 것을 확인할 수 있었다. 고로 데이터를 학습시킬때, 다른 요소보다 단어의 구성 혹은 시퀀스가 더 큰 영향을 끼칠 것이라고 생각해볼 수 있었다.

**Vectorization (벡터화)**

다음으로, 이렇게 토큰화가 완료된 데이터를 실제로 모델이 이해하고 학습할 수 있는 언어로 변경해주는 방법이 벡터화이다. 벡터화란, 문자열 형식의 데이터를 이를 구별할 수 있는 다른 array 혹은 vector값으로 변환하여 모델이 학습할 수 있게 변환시켜 주는 과정이다. Count Based word로 변환한다는 말로도 사용할 수 있다. 이는 어떻게 진행할 수 있을까? 우선 이러한 벡터화에도 여러가지 방법이 존재한다. Bag of Words, Document Term Matrix등의 방식들로 다양한 방법들이 있지만, 이 프로젝트에선 최신방법이자 사용해본 적이 있던 TF-IDF 방식을 사용하였다.

TF-IDF는 위에서 설명한 Document Term Matrix에서 중요도까지 고려한 측정방식이라고 할 수 있다. 이는 단어의 빈도와 역문서 빈도라는 기술을 이용하여 단어마다의 중요정도를 가중치로 주는 방식으로 이 벡터화 과정에서도 fit을 사용해주는 특징이 있다. 여기서 주목해야할 지표는 idf인데 이는 df의 inverse를 뜻하는 과정이다.



이는 쉽게 df값에 inverse를 취해준 후, 데이터 사이즈를 고려한 log를 취한 값으로 확인할 수 있다. 여기서 n이 분자에 있음과 inverse를 두가지를 고려하였을 때, 모든 문서에서 자주 등장하는 단어는 중요도가 낮다는 것을 판단한다는 것을 알 수 있다.

그리고 이러한 벡터화의 특징은 최대 corpus수와 최소 corpus수에 영향을 받는다는 것이다. 모든 단어의 길이를 포함하는 matrix로 변환하여야 하고, 이를 전부 벡터화 시키고 이를 규격에 맞춰야 모두 같은 dimension을 가지는 matrix가 되기 때문에 max corpus를 갖추는 데이터의 corpus를 dimension으로 삼는 matrix가 생성되게 된다. 이를 내가 사용하고자 하는 idf 함수에 하이퍼파라미터로 조정해줄 수 있다. 최대 corpus 길이를 줄이면 데이터의 computational cost를 줄이고 overfitting에 용이함등이 있겠지만 반대로 이는 underfitting혹은 low variance의 problem을 만들 수 있다. 고로 특이한 하이퍼 파라미터 조정은 진행하지 않았다.

이러한 전처리 과정을 통해 학습 모델 선정 이전 데이터 전처리를 완료할 수 있었다.

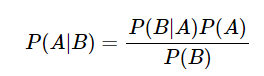
**Model**

**3.1 Model Selection**

모델은 실제 우리가 수업시간에 배웠던 모델들, 그리고 이를 조금더 개선시킨 모델, 마지막으로 가장 좋은 성능을 내는 것으로 유명한 강력한 모델을 선정하기로 하였다. 그래서 naive bayes, tree based, xgboost를 기준으로 모델을 선정하게 되었다.

그리고 수업시간에 추가적으로 배운 LSTM과 같은 딥러닝 기반의 모델도 실제로 구현하여 실행해보면서 모델을 객관적이게 분석하고자 하였다.

**3.1.1 Gaussian Naive Bayes**

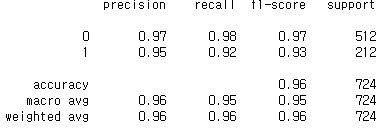
****

우선 Naive Bayes 모델은, 베이즈의 정리(Bayes’ Theorem)을 이용하는 분류방식입니다. 위의 figure가 Bayes’ Theorem인데, 이는 각각의 tokenize된 token들이 각각 스팸메일일 확률을 P(스팸메일ㅣ텍스트)를 이용하여 각각 단어들의 조건부 확률을 계산합니다. 그리고 P(정상메일ㅣ텍스트) 또한 동일하게 진행 합니다. 이는 간단하면서 Computational cost가 적은 점이 가장 큰 장점이 될 수 있습니다. 하지만 이 naive bayes분류는 feature들간에 독립성으로 계산하기 때문에, feature들간의 관계를 계산해주지 않는다는 단점이 있습니다. sklearn의 metrics 라이브러리를 통해 이를 학습하여 보았습니다. 해당 결과로,

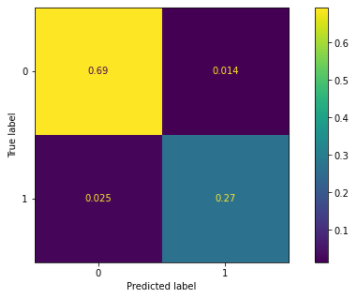
Gaussian Naive Bayes Acc : 0.9613259668508287

Gaussian Naive Bayes Auc : 0.9477815448113207

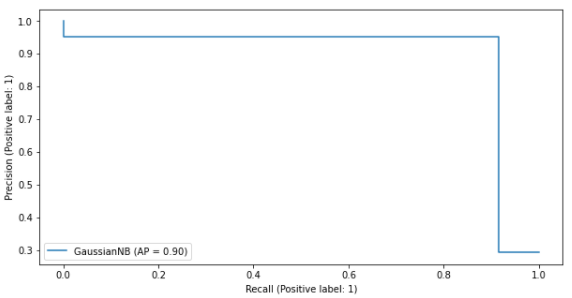
F1-Score, Precision Recall :



Confusion Matrix :



Precision\_Recall\_Curve:

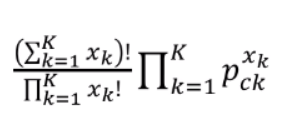


다음의 결과를 얻을 수 있었습니다. 비교적 간단한 모델이라고 가정하고 진행하였고, 큰 기대를 안했지만 생각보다 좋은 정확도와 AUC score를 가졌음을 알 수 있었습니다. 그리고 TP,TN,FP,FN의 수치등을 입력값으로 이용해 표현하는 f1 score와 precision recall를 살펴보아도 높은 수치를 볼 수 있었습니다. 그리고 각각의 인자들을 비교하기 위해 confusion matrix값을 살펴보면 FP, TN 값이 생각보다 아주 적은 수치를 보임을 알 수 있다.

그리고 Precision Recall Curve를 통해 AUC Threshold를 변경하여준 Curve 결과를 살펴보면, 크게 변화가 없음을 통해 비교적 변동없이 학습이 진행되었고, 그럼에도 Curve의 이상적인 그래프와 대략적으로 비슷한 방향으로 학습됨을 확인할 수 있었습니다. 이를 통해 이전의 전처리가 잘 수행되었고, 생각보다 괜찮은 결과값을 보임을 확인할 수 있었습니다.

**3.1.2 Multinomial Naive Bayes**

다음으로, Multinomial Naive Bayes를 진행해보았습니다. 이는 Naive Bayes의 좀 더 발전된 방향을 위해 모델을 선정하게 되었습니다. 위의 첫번째 모델인 Gaussian Naive Bayes는 P(xly) (x는 데이터, y는 스팸 혹은 정상)가 정규분포를 확률분포로 갖고 있기 때문에 앞에 Gaussian이라는 단어가 붙게 된 것이다. 여기서 확률분포를 정규분포가 아닌

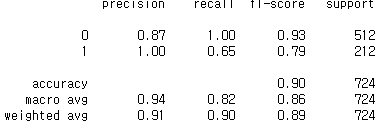


다음의 Multinomial Theorem을 사용해주면 Multinomial Naive Bayes가 되게 되는 것이다. 이는 종속변수는 범주형으로 동일하지만, 설명변수의 형태에 따라 달라짐으로도 생각할 수 있다. GNB는 연속형 설명변수인 반면, MNB는 설명변수가 범주형임으로도 설명이 가능하다. 마찬가지로 sklearn의 metrics를 통해 학습을 진행할 수 있었습니다.

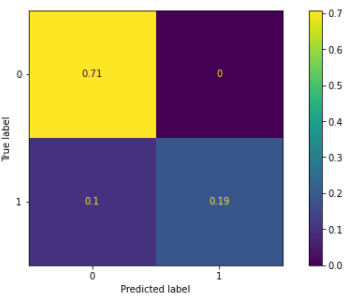
Multinomial Naive Bayes Acc : 0.896408 8397790055

Multinomial Naive Bayes Auc : 0.82311320 75471699

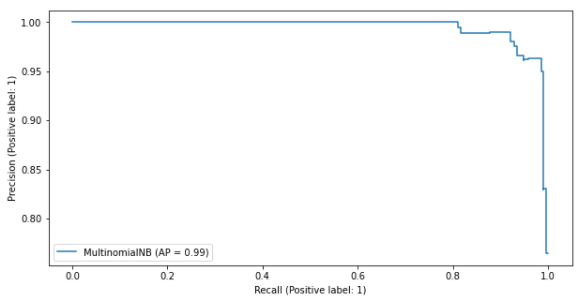
F1-Score, Precision Recall :



Confusion Matrix :



Precision\_Recall\_Curve:



다음 역시 Gaussian Naive Bayes와 비슷한 결과로 AUC, Precision\_Recall\_Curve등에 큰 문제는 발생하지 않음을 확인할 수 있었지만 여전히 Confusion matrix에서 data imbalance가 두드러지게 나타남을 확인할 수 있다 또한 정확도와 AUC Score가 잘 나오지 않음을 확인하였는데, 이는 생각해보기에, 데이터의 distribution이 normal distribution에 더 적합하게 작용한다는 것을 추측해볼 수 있었고, 설명변수가 연속성을 띄고 있음으로 분석해볼 수 있었다.

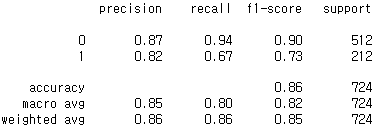
**3.2.1 Decision Tree**

다음으로, Tree 기반의 모델들로 학습을 진행시키기로 하였다. 트리기반의 모델은 기계학습에 가장 잦은 빈도로 사용되는 모델이면서 가장 유명하고 검증된 모델이기도 하다. 우선적으로 Decision Tree (결정 트리) 모델은 데이터의 균일성에 따라 규칙을 기반으로 모델을 학습합니다. 데이터의 균일성을 결정하는 방법으로 Gini Impurity 혹은 information gain 등이 존재합니다. 결정 트리는 노드 혹은 트리가 깊어질 수록 overfitting의 위험이 커지게 되어 학습이 저하될 수 있습니다. 그래서 maximum tree pruning등의 과정을 진행하여 generalization을 높이는 것이 중요한 factor가 되고, 이는 hyperparameter로 조절할 수 있습니다. 이를 우선 학습을 진행하여 보았습니다.

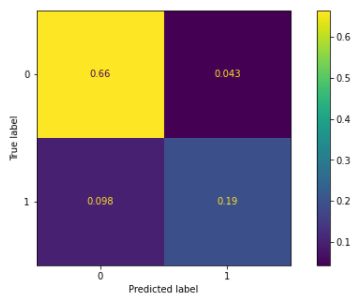
Decision Tree Acc : 0.8591160220994475

Decision Tree Auc : 0.8022737323113207

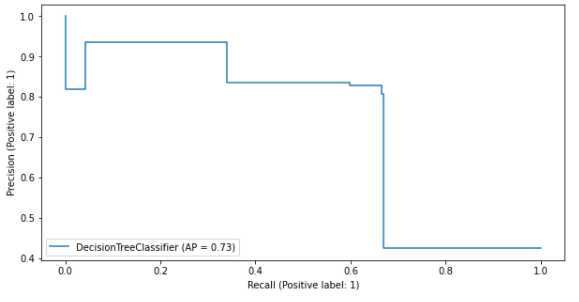
F1-Score, Precision Recall :



Confusion Matrix :



Precision\_Recall\_Curve:



학습결과는 마찬가지로 90을 넘지않은 결과를 보였다. 특히 다른 데이터들에 비해 f1-score에서 특징을 보였는데, 스팸 메일(TN,TP)의 f1-score가 아주 낮게 나타남을 확인하였다. 또한 confusion matrix에서 레이블을 잘 잡지 못한 데이터들 특히 스팸메일을 스팸메일이 아니라고 판단한 경우가 많음을 확인하였다. 이는 모델이 강력하지 않고, underfitting 되었다고도 해석해볼 수 있다. 이러한 결과가 나온 이유는 Hyperparameter를 최선의 하이퍼파라미터로 적용해주지 않음으로도 설명할 수 있고, 다른 이유로는 모델 자체가 강력하지 않은 모델이라고도 해석할 수 있었다. 이를 해결하기 위해 앙상블기반으로 더 강력하고 범용적인 모델인 Random Forest를 사용해보기로 하였다.

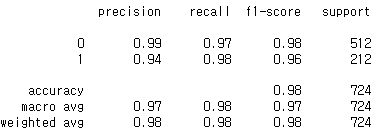
**3.2.2 Random Forest**

우선 random forest 모델을 위해선, 앙상블에 대해 알 필요가 있다. 앙상블은 크게 배깅과 부스팅으로 구분할 수 있는데, 배깅의 경우 각 분류기는 동일한 유형의 알고리즘을 기반으로 서로 다른 데이터 샘플링을 수행하고 학습을 수행하여 보팅을 수행합니다. 또한 개별 분류기에서 데이터를 샘플링하고 추출하는 방법을 부트스트랩 분할 방법이라고 합니다. 여기서 random forest는 배깅 방식의 대표적인 앙상블 모델로 보트를 타고 최종 결정을 내리기 위해 트리 알고리즘을 사용하여 여러 분류기를 만드는 알고리즘입니다. 매개변수에는 n\_estimators라는 의사결정 트리의 수를 지정하는 변수와 의사결정 트리에서 과적합을 개선하는 데 사용되는 max\_feature, max\_depth, min\_samples \_leaf와 같은 매개변수가 포함됩니다.

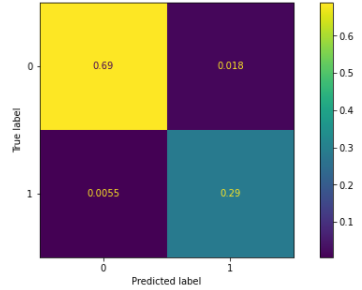
Random Forest Acc : 0.9765193370165746

Random Forest Auc : 0.9778707252358491

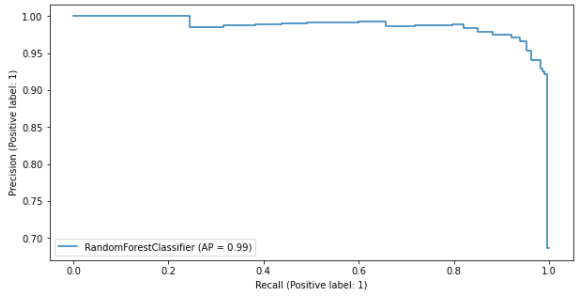
F1-Score, Precision Recall :



Confusion Matrix :



Precision\_Recall\_Curve:



결과를 분석하였을 때, Random Forest가 기대했던 대로 가장 높은 성능을 보임을 확인할 수 있었습니다. 다른 Confusion matrix와 f1-score등을 비교해보아도 가장 높은 성능을 보이고 강력한 모델임을 확인할 수 있었습니다.

하지만, 추가적으로 더 좋은 성능으로 향상 시키기 위해 최적의 하이퍼파라미터를 찾은 Grid Search 방식을 통해 파라미터를 찾아보았는데, 오히려 더 낮은 성능을 보이게 되었습니다.

Grid Parameters :

'n\_estimators' :[5,10,50,100],

'max\_depth' : [6,8,10,12],

'min\_samples\_leaf' : [3,5,7,8,10],

'min\_samples\_split' : [3,6,8,10]

GridSearch RF Acc :0.7679558011049724

GridSearch RF Acc : 0.6079193691037736

이러한 결과가 왜 나왔는지 분석해보았는데, 오히려 더많은 constraints들로 인해 under fitting 되어 더 좋지않은 성능을 보였다가 첫번째 원인이고, 두번째로 파라미터 값을 임의로 지정해주면서, 디폴트 값보다 더 좋지 않은 성능을 갖게되는 하이퍼파라미터들을 grid로 선정하였다 등의 결론을 짓게 되었습니다. 이 과정을 통해 hyperparameter tuning이 얼마나 중요한 task인지, 그리고 같은 모델이라도 다른 파라미터에서 전혀 다른 결과가 나옴을 확인할 수 있었습니다.

**3.3. Others**

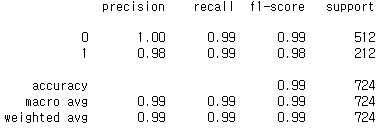
페이지 분량상 더 적을 수 없지만 그 이외에도 Supprot Vectore Machine (Classificer), 그리고 강력한 모델로 알려져있는 XGboost 모델 등을 추가로 진행하였습니다. 여백은 관찰에서 초평면까지의 최단 거리이며 SVM은 분류할 때 이 여백을 최대화하려고 합니다. 즉, 마진을 최대화할 수 있는 경계를 찾는 기술입니다. 결정 경계의 함수에 따라 선형과 비선형으로 구분할 수 있습니다.

그 다음으로 XGboost는 Gradient Boosting 모델에서 Overfitting을 조절하기 위해 하이퍼파라미터와 조기에 멈출 수 있는 파라미터를 추가한 모델입니다. 지정된 반복 횟수가 아닌 교차 검증을 통해 평가 데이터 세트의 평가 값이 최적화되면 중간에 반복을 중지할 수 있는 조기 중단 기능이 있습니다. 강력한 SOTA모델로 알려져 있습니다.

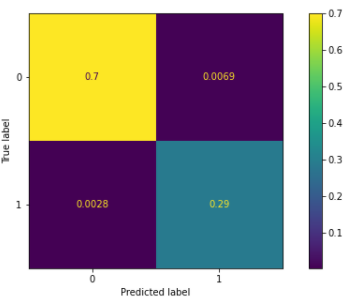
SVC Acc :0.9903314917127072

SVC Acc : 0.9904002063679245

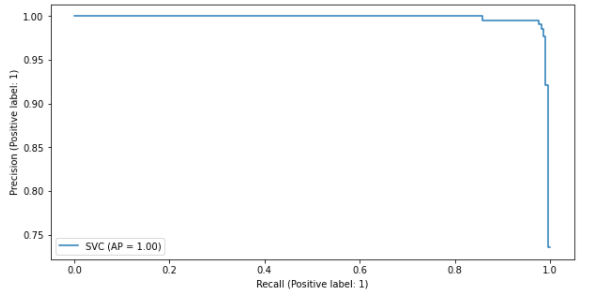
F1-Score, Precision Recall :



Confusion Matrix :



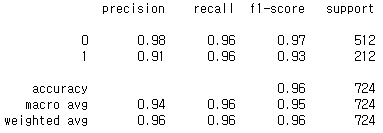
Precision\_Recall\_Curve:



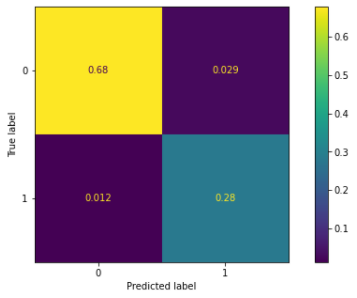
XGBoost Acc : 0.9585635359116023

XGBoost Auc : 0.9582657724056604

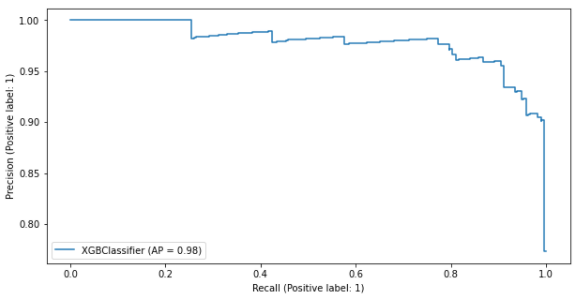
F1-Score, Precision Recall :



Confusion Matrix :



Precision\_Recall\_Curve:



SVC는 99이상의 아주 높은 확률을 보였음을 확인할 수 있었고, XGBoost 역시 높은 성능을 보임을 확인할 수 있었습니다. 다른 자세한 evaluation visualization은 코드를 통해 확인할 수 있습니다.

**3.4. Model comparison**

| Model | AUC score |
| --- | --- |
| Gaussian NB | 0.947782 |
| SVC | 0.990400 |
| Decision Tree | 0.802274 |
| Random Forest | 0.977871 |
| Multinomial NB | 0.823113 |
| XGboost | 0.958266 |

실험을 전부 진행한 결과, SVC가 가장 높은 성능을 보이고, Decision Tree가 가장 낮은 성능을 보이게 되었다.

**3.5. Deep Learning (LSTM)**

마지막으로, 기계학습 뿐만 아니라 딥러닝 특히 수업시간에 배운 LSTM으로 학습을 진행해 보았는데, 가장 좋지 않은 성적을 보이게 되었다. 이는 데이터 전처리 과정 (Cleaning, Tokenize)과정에서 시퀀스 데이터를 잃은 것도 있고, 하이퍼 파라미터와 loss function등을 선정함에 있어서 다른 하이퍼 파라미터를 시행해보지 않아 최적의 모델과 하이퍼 파라미터를 선정하지 못했던 것이 좋지 않은 성적의 결과가 되었던 것 같다. LSTM은 높은 epoch의 딥러닝을 돌렸기 때문에, 따로 GPU 구현이 가능한 서버에서 실행을 돌리고 코드를 옮기는 식으로 학습해보았다.

loss\_func = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(),lr=0. 001)

scheduler = lr\_scheduler.StepLR(optimizer,step\_size =10)

num\_epoch = 10

**Conclusion**

다양한 모델들을 사용해주면서 많은 점을 배우고 실제로 이론으로 배운 모델들이 어떻게 사용되는지 확인할 수 있어서 정말 좋은 경험이었다. 여기서 모델 분석에 있어서 아쉬웠던 점과 개선해야 했던 점을 정리하고자 한다.

**4.1.1 Further Expectation**

우선, Data Preprocessing 과정에서 진행해야 할 부분을 진행하지 못한 지점이 있어서 이에 대해서 이야기 해보고자 한다. Data Cleaning 과정에서 더 고려해주어야 할 점이 있었는데, 영어는 apostrophe라는 문자를 사용하여 텍스트로 표시할때나 이야기할 때 줄이는 경우가 있습니다. 예를 들어, I will은 I’ll로 줄인다던지, I can not을 can’t로 줄이는 등의 표시입니다. 이러한 요소들은 특수문자를 제거할때 제거되어 올바른 언어로 표기될 수 없습니다. 이를 해결해주기 위하여 contractions이라는 라이브러리를 사용해주고자 하였으나, pyahocorasick이라는 라이브러리에 에러가 나서 라이브러리 자체에 에러가 진행되어 위의 진행을 해주지 못하였습니다. 이 과정이 좀 더 원활히 진행되고 이 과정을 포함할 수 있었다면, 더 좋은 성능을 기대할 수 있지 않았을까 라는 생각이 들어, 나중에 추가적으로 이를 진행해보고 싶다.

또한 실제 모델에 학습을 진행시킬 때, 하이퍼 파라미터를 좀 더 다양하게 사용하면 더 좋은 결과를 냈을 것 같은데, 좋은 결과를 내지 못했던 아쉬움이 있는 것 같다. 그래서 추가적으로 성능을 더 높이기 위해 GridSearch와 같은 하이퍼 파라미터 학습 기법을 더 적극적으로 활용할 것이다.